L Number	Hits	Search Text	DB	Time stamp
1	65	(((page) with (flush$3 or invalid$5)) with instruction) same cache	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/12 13:52
2	24	((((page) with (flush$3 or invalid$5)) with instruction) same cache) and (@ad<19980724 or @rlad<19980724) and ((start$3 or begin$4 or range) near5 address)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/12 13:55
3	17	(((((page) with (flush$3 or invalid$5)) with instruction) same cache) and (@ad<19980724 or @rlad<19980724) and ((start$3 or begin$4 or range) near5 address)) and (user or programmer or operator)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/12 13:55

US-PAT-NO: **4648030**

DOCUMENT-IDENTIFIER: US 4648030 A

TITLE: Cache invalidation mechanism for multiprocessor systems

DATE-ISSUED: March 3, 1987

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bomba; Frank C.	Andover	MA	N/A	N/A
Bhandarkar; Dileep P.	Westford	MA	N/A	N/A
Grady, III; J. J.	Woburn	MA	N/A	N/A
Lackey, Jr.; Stanley A.	Littleton	MA	N/A	N/A
Mitchell; Jeffrey W.	Maynard	MA	N/A	N/A
Schumann; Reinhard	Stow	MA	N/A	N/A

ASSIGNEE INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Digital Equipment Corporation	Maynard	MA	N/A	N/A	02

APPL-NO: 06/ 534782

DATE FILED: September 22, 1983

INT-CL: [04] G06F013/00,G06F013/14

US-CL-ISSUED: 364/200

US-CL-CURRENT: **711/141**, 902/39

FIELD-OF-SEARCH: 364/2MSFile; 364/9MSFile

REF-CITED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
3735360	May 1973	Anderson et al.	364/200
	N/A	N/A	
4228503	October 1980	**Waite** et al.	364/200
	N/A	N/A	
4410944	October 1983	Kronies	364/200
	N/A	N/A	
4445174	April 1984	**Fletcher**	364/200
	N/A	N/A	
4481573	November 1984	Fukunaga et al.	364/200
	N/A	N/A	
4503497	March 1985	**Krygowski** et al.	364/200
	N/A	N/A	
4503501	March 1985	Coulson et al.	364/300
	N/A	N/A	
4513367	April 1985	**Chan** et al.	364/200
	N/A	N/A	

OTHER PUBLICATIONS

"A Systematic Approach to the Design of Digital Bussing Structures" by
Kenneth J. Thurber et al., Fall Joint Computer Conference, 1972, pp. 719-740.

ART-UNIT: 232

PRIMARY-EXAMINER: Williams; Archie E.

ATTY-AGENT-FIRM: Cesari and McKenna

ABSTRACT:

One of a plurality of devices on a common communications path (68) has a
local memory (54) that is accessible by other devices on the common
communications path (68). Another device on the common communications path
(68) may include a cache memory (190) that keeps copies of certain of the data

contained by the local memory (54). If another device on the common
communications path (68) accesses the local memory (54), the cache (190) is
kept apprised of this fact by monitoring of the common communications path
(68), and it sets an internal flag to indicate that the data involved may not
be valid. However, the contents of memory 54 may also be accessed by means of
a processor (50) without using the common communications path (68).
Accordingly, provisions are made to send an invalidate signal over the common
communications path (68) when a non-path access of the local memory (54) has
been made to a location to which access was previously afforded over the common
communications path (68). In this way, non-path accesses of a local memory
can be permitted, yet proper invalidation of cache memories can be performed in
a simple manner.

13 Claims, 29 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 17

---------- KWIC ---------

US Patent No. - PN (1):
4648030

Application Filing Date - AD (1):
19830922

Detailed Description Text - DETX (60):
Turning now to FIG. 4B, the Write-type transactions (as implemented, WRITE,
WRITE WITH CACHE INTENT, WRITE MASK WITH CACHE INTENT, and UNLOCK WRITE MASK
WITH CACHE INTENT) are shown in detail. **Starting with the**

Command/Address cycle, the current Master places the appropriate four bit code for the command on information lines I[3:0]; a two-bit code identifying the length of the data transmission on data lines D[31:30]; and an address on data lines D[29:0]. At the same time, it asserts BSY to indicate the occupied status of the communications path, and deasserts NO ARB to signal the availability of the data lines for arbitration during the immediately following cycle.

Detailed Description Text - DETX (65):

When a Write-type transaction occurs on the communications path, devices connected to the path and having resident **cache** memory **invalidate** any cached data within the **address range** of the write **command**. As was the case with the READ WITH CACHE INTENT command, the WRITE WITH CACHE INTENT command, when used with the Invalidate command offers significant performance advantages in certain systems.

Detailed Description Text - DETX (70):

The invalidate transaction is used by systems having cache memories associated therewith. It is issued by devices under certain conditions to guarantee that obsolete data that may be present in the caches of other devices is not used. In the Command/Address cycle of this transaction, as shown in FIG. 4C, the Current Master asserts the Invalidate command on information lines I[3:0] and the **starting address** of the data to be invalidated on data lines D[29:0]. The number of consecutive locations of cached memory to be invalidated is indicated by the data length code on lines D[31:30]. The Command/Address cycle is followed by the usual Imbedded Arbitration cycle, and

a data cycle during which no information is transmitted. As with other
multi-responder commands, the specified permissible responses are ACK and NO
ACK.

Current US Original Classification - CCOR (1):
711/141

US-PAT-NO: **5594876**

DOCUMENT-IDENTIFIER: US 5594876 A

TITLE: Arbitration protocol for a bidirectional bus for handling access requests to a logically divided memory in a multiprocessor system

DATE-ISSUED: January 14, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Getzlaff; Klaus J.	Schoenaich	N/A	N/A	DE
Wille; Udo	Holzgerlingen	N/A	N/A	DE

ASSIGNEE INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY	N/A	N/A	02

APPL-NO: 08/ 514355

DATE FILED: July 27, 1995

PARENT-CASE:

The application is a continuation, of application Ser. No. 08/081,757, filed Jun. 23, 1993, now abandoned.

FOREIGN-APPL-PRIORITY-DATA:

COUNTRY	APPL-NO	APPL-DATE
EP	92110650	June 24, 1992

INT-CL: [06] G06F013/00

US-CL-ISSUED: 395/293, 395/729

US-CL-CURRENT: 710/113, 710/241

FIELD-OF-SEARCH: 395/425; 395/280 ; 395/725

REF-CITED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME
US-CL		
4402040	August 1983	Evett
N/A	**N/A**	N/A
4467419	August 1984	**Wakai**
395/425	N/A	N/A
4473880	September 1984	Budde et al.
N/A	**N/A**	N/A
4621342	November 1986	**Capizzi** et al.
N/A	N/A	N/A
4775955	October 1988	Liu
N/A	**N/A**	N/A
4785394	November 1988	**Fischer**
394/294	N/A	N/A
4851990	July 1989	Johnson et al.
395/325	**N/A**	N/A
4959776	September 1990	**Deerfield** et al.
395/400	N/A	N/A
5060145	October 1991	Scheuneman et al.
395/425	**N/A**	N/A
5133059	July 1992	**Ziegler** et al.
395/425	N/A	N/A
5179708	January 1993	Gyllstrom et al.
395/725	**N/A**	N/A
5222223	June 1993	**Webb,** Jr. et al.
395/425	N/A	N/A
5257356	October 1993	Brockmann et al.
395/290	**N/A**	N/A
5261109	November 1993	**Cadambi** et al.
395/291	N/A	N/A
5276900	January 1994	Schwede
395/800	**N/A**	N/A
5287477	February 1994	**Johnson** et al.
395/425	N/A	N/A

5321697	June 1994	Fromm et al.
371/10.1	**N/A**	N/A
5323489	June 1994	**Bird**
395/425	N/A	N/A
5379379	January 1995	Becker et al.
395/425	**N/A**	N/A
5490253	February 1996	**Laha** et al.
395/304	N/A	N/A

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY
US-CL		
0054888	June 1982	EP
0301921	January 1989	EP
0379436	July 1990	EP
63-91757	April 1988	JP

ART-UNIT: 235

PRIMARY-EXAMINER: Harvey; Jack B.

ASSISTANT-EXAMINER: Wiley; David A.

ATTY-AGENT-FIRM: Jenkens & Gilchrist Shkurko; Eugene I.

ABSTRACT:

The invention concerns a multiprocessor system comprising processors PU0 to PUn and a common main memory. The memory is logically divided into at least two banks M0 and M1 and is interconnected with the processors by a bus 110. By means of control lines 111 to 118 a bus protocol is established so that one of said memory banks is accessed while another one of said banks is still busy.

9 Claims, 14 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 13

---------- KWIC ---------

US Patent No. - PN (1):
5594876

Application Filing Date - AD (1):
19950727

Brief Summary Text - BSTX (13):
Another alternative is to have software control of the coherence of certain data for which efficient implementation of castout control may be constrained. The idea is to flush lines out of the private caches when there is a danger of data pollution through stores from other processors. Such pollution may occur when, for example, an object is released by a task running on a processor and hence tasks on other processors may obtain the resource and modify it. In some computer architectures there are instructions offered to flush data lines out of the cache. Such **cache flush instructions** are designed by specifying **address ranges** in which lines are to be replaced from the **cache**. Such approaches force the software, e.g. the compiler or the programmer, to keep track of the addressed ranges for flushing. **Address range** is a non-semantic specification of logical objects in software. Therefore, such cache flushing instructions make storage systems less transparent to software.

Related Application Filing Date - RLFD (1):
19930623

US-PAT-NO: **5768593**

DOCUMENT-IDENTIFIER: US 5768593 A

TITLE: Dynamic cross-compilation system and method

DATE-ISSUED: June 16, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Walters; Chad Perry	Redwood City	CA	N/A	N/A
Brown; Jorg Anthony	Concord	CA	N/A	N/A

ASSIGNEE INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Connectix Corporation	San Mateo	CA	N/A	N/A	02

APPL-NO: 08/ 620387

DATE FILED: March 22, 1996

INT-CL: [06] G06F009/30,G06F009/44

US-CL-ISSUED: 395/705, 395/707 , 395/709 , 395/500 , 395/581

US-CL-CURRENT: 717/141, 703/26 , 712/234

FIELD-OF-SEARCH: 395/700; 395/701-710 ; 395/712 ; 395/500 ; 395/581

REF-CITED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4667290	May 1987	Goss et al.	

	395/707	**N/A**	N/A
5167023		November 1992	**De** Nicolas et al.
	395/527	N/A	N/A
5179703		January 1993	Evans
	395/703	**N/A**	N/A
5204960		April 1993	**Smith** et al.
	395/707	N/A	N/A
5367683		November 1994	Brett
	395/709	**N/A**	N/A
5625822		April 1997	**Brett**
	395/705	N/A	N/A
5649203		July 1997	Sites
	395/709	**N/A**	N/A
5692196		November 1997	**Unni** et al.
	395/705	N/A	N/A

ART-UNIT: 274

PRIMARY-EXAMINER: Voeltz; Emanuel Todd

ASSISTANT-EXAMINER: Dam; Tuan Q.

ATTY-AGENT-FIRM: Williams; Gary S. Flehr Hohbach Test Albritton & Herbert LLP

ABSTRACT:

In a computer system, a cross-compiler converts non-native code into native code immediately prior to execution of that code. The system also includes a code cache for storing cross-compiled code and a hash table for locating code blocks in the code cache. In a preferred embodiment, the system also includes an interpreter for emulating certain non-native instructions that are not converted into native code by the cross-compiler. While executing any non-native application, if the next instruction is not one of the predefined set of non-native instructions to be handled by interpretation or a special purpose procedure, then the next instruction is considered to be an "entry point" instruction, and the cross-compiler looks up the

address of the entry
point instruction in the hash table to see if a corresponding native code block
is already stored in the code cache. If so, the native code block in the code
cache is executed until an exit instruction in the native code block is
encountered. Otherwise, the cross-compiler cross-compiles all code that is
reachable from the entry point instruction during execution of the program
without going outside the compilation window. During compilation the
cross-compiler determines the non-native condition codes generated by a
non-native instruction that will not be used by any successors of the
non-native instruction. The native code instructions generated by the
cross-compiler do not include instructions for processing non-native condition
codes generated by the non-native instruction that will not be used by any
successors of the qualifying non-native instruction.

12 Claims, 6 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 5

---------- KWIC ---------

US Patent No. - PN (1):
5768593

Application Filing Date - AD (1):
19960322

Brief Summary Text - BSTX (23):
Whenever a **partial cache flush instruction** is executed, the specified
portion of the system's **cache** memory is cleared, and any corresponding entries

in the hash table and code chunk map are also cleared. More particularly, the
code chunk map entries corresponding to the flushed **address range** are
inspected, and for each code chunk map entry that is set a corresponding
portion of the hash table is cleared so as to prevent use of the corresponding
code blocks in the code cache. This is more efficient than simply invalidating
all code blocks in the code cache because it allows much of the previously
cross-compiled code in the code cache to continue to be used.

Detailed Description Text - DETX (28):
Whenever a non-native **partial cache flush instruction** is found in a code
block that the recompiler is compiling, the **partial cache flush instruction** is
recompiled into a procedure call to a special purpose **partial cache flush**
procedure 146. That procedure 146 inspects the code chunk map 180 entries for
the **address range flushed from the cache,** and for each code chunk map entry
that is set a corresponding **portion** of the hash table is searched, and all
entries for non-native entry point **instructions in that portion** of the hash
table are cleared so as to prevent use of the corresponding code blocks in the
code **cache**. Because the hash function 144 used by the recompiler is a linear
function, the portion of the hash table 122 that corresponds to any entry in
the code chunk map 180 is easily determined. In particular, all hash table
entries between HashTableBeginClear and HashTableEndClear are cleared, where

Claims Text - CLTX (19):
said cross-compiling step further including generating for each said
qualifying **instruction** composing a **partial cache flush instruction** a native

code **instruction** that invokes a predefined **partial cache flush** procedure, said predefined **partial cache flush** procedure clearing a specified **address range** from said computer system's **cache** memory, inspecting the code chunk map entries corresponding to the specified **address range flushed from the cache** memory, and for each such code chunk map entry that indicates said code **cache** contains a code block corresponding to a non-native entry point **instruction** in said associated **page,** clearing a corresponding **portion** of the lookup table so as to prevent use of the corresponding code blocks in the code **cache**.

Claims Text - CLTX (38):
said cross-compiling procedure further including **instructions** for generating for each said qualifying **instruction** composing a **partial cache flush instruction** a native code **instruction** that invokes a predefined **partial cache flush** procedure, said predefined **partial cache flush** procedure clearing a specified **address range** from said computer system's **cache** memory, inspecting the code chunk map entries corresponding to the specified **address range flushed from the cache** memory, and for each such code chunk map entry that indicates said code **cache** contains a code block corresponding to a non-native entry point **instruction** in said associated **page,** clearing a corresponding **portion** of the lookup table so as to prevent use of the corresponding code blocks in the code **cache**.

US-PAT-NO: **5524233**

DOCUMENT-IDENTIFIER: US 5524233 A
See image for Certificate of Correction

TITLE: Method and apparatus for controlling an external cache memory wherein the cache controller is responsive to an interagent communication for performing cache control operations

DATE-ISSUED: June 4, 1996

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Milburn; Blair D.	Beaverton	OR	N/A	N/A
Lee; Phillip G.	Aloha	OR	N/A	N/A
Karnik; Milind A.	Aloha	OR	N/A	N/A

ASSIGNEE INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Intel Corporation	Santa Clara	CA	N/A	N/A	02

APPL-NO: 08/ 040680

DATE FILED: March 31, 1993

INT-CL: [06] G06F013/00

US-CL-ISSUED: 395/468, 395/449 , 395/451 , 395/460 , 395/462 , 364/243.41 , 364/243.45 , 364/964.2 , 364/964.343 , 364/DIG.1 , 364/DIG.2

US-CL-CURRENT: **711/141**, 711/122 , 711/124 , **711/133** , 711/135

FIELD-OF-SEARCH: 364/243.45; 395/427 ; 395/445 ; 395/446 ; 395/448 ; 395/449
; 395/451 ; 395/460 ; 395/462 ; 395/468 ; 395/473 ; 395/474

REF-CITED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4829425	May 1989	Bain, Jr. et al.	395/275
	N/A	N/A	
5214770	May 1993	**Ramanujan** et al.	395/425
	N/A	N/A	
5241681	August 1993	Hamid et al.	395/800
	N/A	N/A	
5247643	September 1993	**Shottan**	395/425
	N/A	N/A	
5276848	January 1994	Gallagher et al.	395/425
	N/A	N/A	
5287523	February 1994	**Allison** et al.	395/725
	N/A	N/A	
5293384	March 1994	Keeley et al.	371/16.3
	N/A	N/A	
5325499	June 1994	**Kummer** et al.	395/425
	N/A	N/A	
5345576	September 1994	Lee et al.	395/425
	N/A	N/A	
5347648	September 1994	**Stamm** et al.	395/575
	N/A	N/A	
5394529	February 1995	Brown, III et al.	395/375
	N/A	N/A	
5398325	March 1995	**Chang** et al.	395/425
	N/A	N/A	

ART-UNIT: 238

PRIMARY-EXAMINER: Swann; Tod R.

ASSISTANT-EXAMINER: Asta; Frank J.

ATTY-AGENT-FIRM: Blakely, Sokoloff, Taylor & Zafman

ABSTRACT:

A cache control method and mechanism for an external cache memory having
multiple cache lines using interagent communications to cause invalidating the
external cache memory, flushing the external cache memory and/or changing the
coherency state of lines in the external cache memory.

43 Claims, 10 Drawing figures

Exemplary Claim Number: 32

Number of Drawing Sheets: 10

---------- KWIC ---------

US Patent No. - PN (1):
5524233

Application Filing Date - AD (1):
19930331

Detailed Description Text - DETX (43):
Field3 and Field4 provide the **address range** upon which the operation
specified in the control bits are to be performed. By being able to specify an
address range instead of a single cache line address, the present invention can
flush, modify the coherency state, or **invalidate portions** or all of external
cache using one **command**.

Detailed Description Text - DETX (44):
In the present invention, Field3 and Field4 are aligned on quad (16-byte)
boundaries. In the currently preferred embodiment, if a non-aligned **starting**
address is used, then bytes in the quad before the starting address may also be
invalidated. In the currently preferred embodiment, if a

non-aligned ending
address is used, then bytes in the quad containing that address may not be
invalidated. To invalidate one 16-byte line in an external cache, both Field3
and Field4 should be specified as the physical address of the line.

Detailed Description Text - DETX (75):
Next, the processor is set to use physical addressing for accesses
(processing block 711). Then the **starting address** in the IAC message is stored
into the variable (LOAD.sub.-- ADDR) (processing block 712). Next, a loop is
entered in which the four words **starting at the address** in the variable
LOAD.sub.-- ADDR are loaded and stored into a temporary register (processing
block 713).

Detailed Description Text - DETX (83):
Thus, by using the microcode routine above, in conjunction with the cache
bus control logic, the present invention allows for the system and user
software to manipulate an external cache. This manipulation could occur during
system reset of the external cache. In other embodiments, this manipulation
could occur when it is desired to place the cache in a known state. Note also
that the software does not require knowledge of the cache organization to
accomplish the functions desired (e.g., flushing invalidating, etc.). For
example, the present invention provides the ability to invalidate an external
cache based on an **address range** (as specified in the IAC message). That is,
since only a **range of addresses are specified in the IAC message and the
address range** specified is not dependent on the size or organization of the
external cache, the software is only dependent on the

processor performing the
routine. Therefore, the software utilizing the present invention could easily
be ported to hardware systems which have varying sized caches.

Detailed Description Paragraph Table - DETL (1):
Msg. Type: 9B HEX
Parameters: Field2
Contains control bits: bit 0 0 (flush) or 1 (modify coherency state) bit
1 0 (data cache) or 1 (instruction cache) bit 2 0 (enable match) or 1
(disable match) bits 6:7 coherency state Field3
Quad-aligned **starting physical address** of the flush Field4 Quad-aligned ending physical address of
the flush. The addressed quad is the last quad flushed or modified.

Claims Text - CLTX (9):
7. The processor as defined in claim 1 wherein the cache controlling means
performs at least one operation on a plurality of lines of the cache based on
an **address range** specified in each of the interagent communication.

Claims Text - CLTX (33):
20. The processor as defined in claim 17 wherein the interagent
communication further includes means for specifying an **address range,** such that
the cache control logic performs cache control operations on the **address range**
in the external cache according to the setting of each of the plurality of
control bits.

Claims Text - CLTX (34):
21. The processor as defined in claim 20 wherein the means for specifying

an **address range comprises a starting address field for storing the starting address of the address range and an ending address** field for storing the ending **address of the address range**.

Claims Text - CLTX (56):
and wherein the interagent communication comprises a plurality of bits for specifying at least one cache control operation and an **address range,** such that at least one of the lines in the cache memory is flushed, invalidated and/or its coherency state modified by the first processor according to the plurality of bits and the **address range** in the interagent communication.

Claims Text - CLTX (58):
said first processor sending a interagent communication, wherein the interagent communication includes a plurality of bits for specifying at least one cache control operation and an **address range**;

Current US Original Classification - CCOR (1):
711/141

Current US Cross Reference Classification - CCXR (3):
711/133

US-PAT-NO: **4775955**

DOCUMENT-IDENTIFIER: US 4775955 A

TITLE: Cache coherence mechanism based on locking

DATE-ISSUED: October 4, 1988

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Liu; Lishing	Millwood	NY	N/A	N/A

ASSIGNEE INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY	N/A	N/A	02

APPL-NO: 07/ 134676

DATE FILED: December 18, 1987

PARENT-CASE:

This application is a continuation of application Ser. No. 792,728 filed Oct. 30, 1985 now abandoned.

INT-CL: [04] G06F009/00

US-CL-ISSUED: 364/900

US-CL-CURRENT: 711/145, 711/135

FIELD-OF-SEARCH: 364/200; 364/900

REF-CITED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4322795	March 1982	Lange et al.	364/200
	N/A	N/A	
4394731	July 1983	**Flusche** et al.	364/200
	N/A	N/A	
4426681	January 1984	Bacot et al.	364/200
	N/A	N/A	
4471429	September 1984	**Porter** et al.	364/200
	N/A	N/A	
4484267	November 1984	Fletcher	364/200
	N/A	N/A	
4493026	January 1985	**Olnowich**	364/200
	N/A	N/A	
4513367	April 1985	Chan et al.	364/200
	N/A	N/A	
4620275	October 1986	**Wallach** et al.	364/200
	N/A	N/A	
4622631	November 1986	Frank et al.	364/200
	N/A	N/A	

ART-UNIT: 237

PRIMARY-EXAMINER: Shaw; Gareth D.

ASSISTANT-EXAMINER: Mills; John Gladstone

ATTY-AGENT-FIRM: Arnold; Jack M. Ilardi; Terry J.

ABSTRACT:

A method and apparatus is provided for associating in cache directories the Control Domain Identifications (CDIDs) of software covered by each cache line. Through the use of such provision and/or the addition of Identifications of users actively using lines, cache coherence of certain data is controlled without performing conventional Cross-Interrogates (XIs), if the accesses to such objects are properly synchronized with locking type concurrency controls. Software protocols to caches are provided for the resource kernel to control the flushing of released cache lines. The parameters of

these protocols are
high level Domain Identifications and Task Identifications.

4 Claims, 7 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 6

---------- KWIC ---------

US Patent No. - PN (1):
4775955

Application Filing Date - AD (1):
19871218

Brief Summary Text - BSTX (9):
Another alternative is to have software control the coherence of certain
data for which efficient implementation of XI control may be constrained. The
idea is to flush lines out of the private cache when there is danger of data
pollution through stores from other CPs. Such pollution may occur when, for
example, an object is released by a task running on the CP, and hence tasks on
other CPs may obtain the resource and modify it. In some computer
architectures (e.g., the IBM 801) there are instructions offered to flush data
lines out of a cache. Such **cache flush instructions** are designed by specifying
address ranges in which lines are to be replaced from the **cache**. Such
approaches force the software (e.g., the compiler or the programmer) to keep
track of the **address ranges** for flushing. **Address range** is a non-semantic
specification of logical objects in software. Therefore, such cache flushing
instructions make storage systems less transparent to

software.

Detailed Description Text - DETX (6):

Assume that the MS is partitioned into BLOCKs of size a power of 2 (e.g., 256 bytes). The BLOCK size should be at least as big as the sizes of all cache lines involved. Hence a line cannot be separated into two BLOCKs. Objects with block disjoint property are being considered. An object has block disjoint property if, at any instance of time, the MS blocks allocated to contain this object should not contain any accessible data not belonging to the object. As far as cache coherence is concerned, one has to make sure that a line in a cache is kicked out of the cache when there is any potential for this line to be modified by other CPs and that modifications to a line in cache is always updated to MS before this line can be accessed from other CPs. These conditions may be met when lines belonging to a block are flushed out of the cache (with modified lines updated to the MS) as soon as there is no more active user on the CP still holding lock on the object covering this block.

There have been architectures, for example the IBM 801 computer, in which instructions are provided to flush lines out of the cache. However, such **instructions for cache** line **flushing use address ranges** to specify the lines to be **flushed**. This puts a burden on the software to keep track of the **address ranges** of software objects, which also makes the storage system less transparent to the software.

Related Application Filing Date - RLFD (1):

19851030

US-PAT-NO: **5539895**

DOCUMENT-IDENTIFIER: US 5539895 A

TITLE: Hierarchical computer cache system

DATE-ISSUED: July 23, 1996

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bishop; James W.	Endicott	NY	N/A	N/A
Carmack, Jr.; Charles E.	Rochester	MN	N/A	N/A
Gallagher; Patrick W.	Rochester	MN	N/A	N/A
Jackowski; Stefan P.	Endicott	NY	N/A	N/A
Klouda; Gregory R.	Endwell	NY	N/A	N/A
Siegl; Robert D.	Endicott	NY	N/A	N/A

ASSIGNEE INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY	N/A	N/A	02

APPL-NO: 08/ 241910

DATE FILED: May 12, 1994

INT-CL: [06] G06F012/00,G06F012/08 ,G06F012/12 ,G06F013/00

US-CL-ISSUED: 395/465, 395/444 , 395/445 , 395/446 , 395/447 , 395/449 , 395/468 , 395/492

US-CL-CURRENT: 711/138, 711/117 , **711/118** , 711/119 , 711/120 , 711/122

, **711/141 , 711/165**

FIELD-OF-SEARCH: 395/465; 395/449 ; 395/451 ; 395/466 ; 395/445 ; 395/446
; 395/473 ; 395/468 ; 395/444 ; 395/447 ; 395/492

REF-CITED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
3735360	May 1973	Anderson et al.	395/250
	N/A	N/A	
4044337	August 1977	**Hicks** et al.	395/250
	N/A	N/A	
4084231	April 1978	Capozzi et al.	395/444
	N/A	N/A	
4096567	June 1978	**Millard** et al.	395/600
	N/A	N/A	
4298929	November 1981	Capozzi	395/411
	N/A	N/A	
4394731	July 1983	**Flusche** et al.	395/472
	N/A	N/A	
4394733	July 1983	Swenson	395/403
	N/A	N/A	
4442487	April 1984	**Fletcher** et al.	395/449
	N/A	N/A	
4445174	April 1984	Fletcher	395/448
	N/A	N/A	
4467411	August 1984	**Fry** et al.	395/250
	N/A	N/A	
4484267	November 1984	Fletcher	395/451
	N/A	N/A	
4525780	June 1985	**Bratt** et al.	395/490
	N/A	N/A	
4535455	August 1985	Peterson	395/182.04
	N/A	N/A	
4564899	January 1986	**Holly** et al.	395/284
	N/A	N/A	
4633440	December 1986	Palulski	365/189.05
	N/A	N/A	
4703481	October 1987	**Fremont**	395/182.15
	N/A	N/A	

4774654		September 1988	Pomerene et al.
	395/449	**N/A**	N/A
4797814		January 1989	**Brenza**
	395/403	N/A	N/A
4891809		January 1990	Hazawa
	371/3	**N/A**	N/A
4907228		March 1990	**Bruckert** et al.
	395/182.09	N/A	N/A
4912707		March 1990	Kogge et al.
	395/182.15	**N/A**	N/A
4924466		May 1990	**Gregor** et al.
	395/182.14	N/A	N/A
4947319		August 1990	Bozman
	363/134	**N/A**	N/A
5025366		June 1991	**Baror**
	395/455	N/A	N/A
5097532		March 1992	Borup et al.
	395/425	**N/A**	N/A
5113514		May 1992	**Albonesi** et al.
	395/471	N/A	N/A
5153881		October 1992	Bruckert et al.
	395/182.08	**N/A**	N/A
5155832		October 1992	**Hunt**
	395/447	N/A	N/A
5265212		November 1993	Bruce
	395/325	**N/A**	N/A
5325503		June 1994	**Stevens** et al.
	395/425	N/A	N/A
5341487		August 1994	Derwin et al.
	395/425	**N/A**	N/A
5353423		October 1994	**Hamid** et al.
	395/425	N/A	N/A
5355467		October 1994	MacWilliams et al.
	395/425	**N/A**	N/A
5388246		February 1995	**Kasai**
	395/425	N/A	N/A
5426765		June 1995	Stevens et al.
	395/425	**N/A**	N/A
5446863		August 1995	**Stevens** et al.
	395/427	N/A	N/A

OTHER PUBLICATIONS

Shared Memory Systems on the Futurebus, Paul Sweazey, COMPCON Spring 88
IEEE Computer Society Intl Conference.

Dec 7000/10000 Model 600 AXP Multiprocessor Server, Brian Allison, COMPCON
Spring '93, IEEE Computer Society Intl Conference.

IBM Technical Disclosure Bulletin, vol. 34, No. 3, Aug. 1991, pp. 256-258.

ART-UNIT: 238

PRIMARY-EXAMINER: Swann; Tod R.

ASSISTANT-EXAMINER: Chow; Christopher S.

ATTY-AGENT-FIRM: Samodovitz; Arthur J.

ABSTRACT:

A hierarchical cache system comprises a plurality of first level cache subsystems for storing data or instructions of respective CPUs, a higher level cache subsystem containing data or instructions of the plurality of cache subsystems, and a main memory coupled to the higher level cache subsystem. A page mover is coupled to the higher level cache subsystem and main memory, and responds to a request from one of the CPUs to store data into the main memory, by storing the data into the main memory without copying previous contents of a store-to address of the request to the higher level cache subsystem in response to said request. Also, the page mover invalidates the previous contents in the higher level cache subsystem if already resident there when the CPU made the request. A buffering system within the page mover comprises request buffers and data segment buffers to store a segment of predetermined size of the data. When all of the request buffers have like priority and there are fewer request buffers that contain respective, outstanding requests than the number of data segment buffers, the page mover means allocates to the request buffers with

outstanding requests use of the data segment buffers for which there are no
outstanding requests.

18 Claims, 27 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 21

---------- KWIC ---------

US Patent No. - PN (1):
5539895

Application Filing Date - AD (1):
19940512

Detailed Description Text - DETX (14):
Each of the I/O processors 53a-d (FIG. 1) can also issue the following two
commands. I/O PAGE COPY means that the page mover should read data from the 4K
From address and store the data into the 4K To address. If the data resides in
the L2 cache, it is read from the L2 cache but not invalidated in the L2 cache
From address. Then, the data is stored to the destination and the L2 copy is
invalidated at the L2 cache To address. I/O **PAGE** STORE ZERO **command means that**
the page mover should store all zeros in the 4K To address, and **invalidate** any
copy in the L2 **cache** at the L2 **cache** To address.

Detailed Description Text - DETX (15):
FIG. 4 illustrates flow of a CPU PAGE STORE request from CPU 22a to update
data in main memory (step 110). The CPU 22a sends the request to the traffic
cop 50a (FIG. 2) via L1 interface 54a (step 111). The request specifies a page

of data to be written, the To **address, i.e. the starting address** for storage of the page of data, and that the data should be stored in main memory 51. (Alternately, the request could specify an address in extended memory, to store to.) Because the request is a main memory request, traffic cop 50a sends the request to traffic cop 70a of storage controller 38a via SC interface 85a and L20 interface 74a. The traffic cop 70a receives the CPU PAGE STORE request and then prioritizes it behind previously received requests of all types from the CPU. The traffic cop 70a also prioritizes ahead of all CPU requests all types of I/O processor requests whether previously received or subsequently received (before the remaining CPU requests are executed). Therefore, the traffic cop 70a sends any I/O processor requests to the page mover before any CPU requests, and in the absence of any I/O processor requests sends the CPU requests in FIFO order to the page mover (step 112).

Detailed Description Text - DETX (20):

Referring again to the affirmative output of decision 122, while page mover executes steps 124, 126 etc., the page mover also increments the previous line address by one hundred twenty eight bytes to identify the **starting address** of the next line of the page (step 140). Then, the page mover determines if the result indicates an end of a page (decision 142). If so, then the page mover stops the incrementing. However, if the result does not indicate the end of a page, the page mover determines if a key check is required (step 144). A key check is required if the **address indicates the start** of a new page or if the key for the page was changed since the last check. If a key check is required, the page mover loops back to step 162 to check the key in the manner noted

above.

Detailed Description Text - DETX (21):
The following is a flow description for an I/O PAGE COPY request by I/O processor 53a (FIG. 1) as illustrated in FIG. 5. The I/O processor 53a sends the request to the traffic cop 70a via I/O processor interface 97a (or 99a) (step 150). The request specifies the **starting From address** of the page in main memory 51 (FIG. 1) to be copied from and the **starting To address** in main memory to copy to. (Alternately, the request could specify an address in extended memory or L3 cache, if there was one, to copy from or to.) Then, traffic cop 70a passes the request, after all previously received I/O processor requests, to page mover 71a for processing (step 152). In response, the page mover determines if one of the request buffers 100a-d is available (decision 156). If not, the page mover sets a busy bit, indicating a busy response was sent to the requesting I/O processor, and returns a busy response to the traffic cop 70a and the traffic cop 70a forwards the busy response to the requesting I/O processor (step 158). If one of the request buffers is available, the page mover loads the request into it and returns an acceptance signal to the requesting I/O processor via the traffic cop 70a (step 160).
Next, the page mover requests the key for the From address from the main memory key array 49 (step 162), and compares this key to the key of the requesting I/O processor. If the I/O processor does not have the requisite authority, the page mover returns a key error signal to the traffic cop 70a and the traffic cop passes the key error signal to the I/O processor (decision 164 and step 166). However, if the I/O processor has authority to access the address

specified in the request buffer, the page mover next determines if the associated line buffer 102a is free (decision 172). If not, the page mover determines if another one of the line buffers 102b-d is available to I/O processor 53a (decision 174). If so, or if the answer to decision 172 was yes, then the page mover puts a line fetch request portion of the I/O PAGE COPY REQUEST onto execution stack 104 (step 176). When this line request is next on execution stack 104 (decision 177), the page mover passes the line fetch request to the traffic cop 70a (step 178) and then the traffic cop 70a fetches the line of data from main memory 51 (step 180). (However, if the data also resides in the L2 cache or in the modified line buffer, then the traffic cop 70a fetches the data from the L2 cache or modified line buffer instead.) Next, the traffic cop 70a returns the data to the page mover and the page mover loads the data in the line buffer (step 181). Next, the page mover determines if a key check is required to write to the desired memory locations (decision 182). If the line in the line buffer is the first line of a page or if the key for the To address has been changed, then the I/O processor's key must be checked. This check is performed by fetching the key for this page from key array 49 (step 184), and then comparing the I/O processor's key provided with the request for this page to the key fetched from main memory (step 186). If the I/O processor is not authorized to write to the address specified in the request, the page mover returns a key error signal to the traffic cop 70a, and the traffic cop 70a returns the key error signal to the I/O processor (step 187). Otherwise, the page mover next puts a store request corresponding to the line buffer that just completed the fetch onto execution stack 104 (FIG. 3)

(step 188). As noted above, the I/O operations have higher priority than the CPU operations (decision 189), and when this line store request reaches to the top of the stack 104, the page mover sends it to the traffic cop 70a and the traffic cop 70a sends the line of data from the line buffer to the modified line buffer (steps 192 and 194). While the data is received into the modified line buffer, the traffic cop 70a checks the copy directory in the storage controller to determine if the L2 cache contains a copy of this line (step 195). If so, the traffic cop 70a sends an invalidate signal to the traffic cop 50a of the L2 cache subsystem 31a for this line and the invalidate signals is stored in the L2 directory 34a (step 196). While the traffic cop 50a invalidates the copy in the L2 cache, the L2 controller also checks its L1 copy directories 54a-d to determine if any of the L1 caches contain a copy of this page (decision 197), and if so, sends an invalidate signal to the L1 directory or directories in the L1 cache subsystems that contain a copy of this line (step 198). After sending the line to the modified line buffer and the invalidating the L2 and L1 cache copies, the page mover frees the line buffer (step 199).

Detailed Description Text - DETX (24):

Referring again to the affirmative output of decision 172, while page mover executes 176, 178 etc., the page mover also increments the previous line address by one hundred twenty eight bytes to identify the **starting address** of the next line of the page (step 210). Then, the page mover determines if the result indicate an end of a page (decision 212). If so, then the page mover stops the incrementing. However, if the result does not indicate the end of a

page, the page mover determines if a key check is required (step 214). A key check is required if the address indicates the start of a new page or if the key for the page was changed since the last check. If a key check is required, the page mover loops back to step 162 to check the key in the manner noted above.

Detailed Description Text - DETX (27):

The CPU PAGE COPY & PURGE operation is the same as the I/O PAGE COPY operation and is executed as in FIG. 5 except the CPU PAGE COPY & PURGE command originates from a CPU instead of an I/O processor and during the fetch from L2 cache (steps 180-181, assuming the data resides in the L2 cache), the CPU PAGE COPY & PURGE operation always invalidates the From address in the L2 cache of the L2 copy that was just fetched whereas the CPU PAGE COPY operation does not invalidate the From address in the L2 cache of the L2 copy upon the fetch operation. During the subsequent store operation for the CPU PAGE COPY & PURGE, CPU PAGE COPY and I/O PAGE COPY operations, if the data for the To address also resides in the L2 cache it is invalidated there.

Current US Cross Reference Classification - CCXR (2):

711/118

Current US Cross Reference Classification - CCXR (6):

711/141